Coq Modulo Theory

Pierre-Yves Strub

Coq Modulo Theory

Pierre-Yves Strub

INRIA - Tsinghua University
Offices 3-604, FIT Building, Tsinghua University
Haidian District, Beijing, 100084, CHINA

Abstract. Coq Modulo Theory (CoqMT) is an extension of the Coq proof assistant incorporating, in its computational mechanism, validity entailment for user-defined first-order equational theories. Such a mechanism strictly enriches the system (more terms are typable), eases the use of dependent types and provides more automation during the development of proofs.
CoqMT improves over the Calculus of Congruent Inductive Constructions by getting rid of various restrictions and simplifying the type-checking algorithm and the integration of first-order decision procedures.
We present here CoqMT, and outline its meta-theoretical study. We also give a brief description of our CoqMT implementation.

1 Introduction

Theorem provers like Coq [1] based on the Curry-Howard isomorphism enjoy a mechanism which incorporates computations within deductions. This allows replacing the proof of a proposition by the proof of an equivalent proposition obtained from the former thanks to possibly complex computations. Adding more power to this mechanism leads to a calculus which is more expressive (more terms are typable), which provides more automation (more deduction steps are hidden in computations) and eases the use of dependent data types in proof development.

Coq was initially based on the Calculus of Constructions (CC) of Coquand and Huet [2], which is an impredicative type theory incorporating polymorphism, dependent types and type constructors. At that time, computations were restricted to β-reduction: other forms of computations were encoded as deductions.

The Calculus of Inductive Constructions of Coquand and Paulin [3–5] introduced inductive types and their associated elimination rules. CIC allows for example the definition of Peano natural numbers based on the two constructors 0 and S along with the ability to define addition by induction: $x + 0 \to x$ and $x + S(y) \to S(x + y)$. This mechanism allows the system to identify the expressions $x + S(S(y))$ and $S(x + y) + S(0)$, but fails in identifying $x + S(y)$ and

[1] This work was partly supported by the ANR ANR-08-BLAN-0326-01

$S(x) + y$. This forbids users to easily define functions on dependent data-types (like the `reverse` function on lists) as the types $\mathbf{list}(n + 1)$ and $\mathbf{list}(1 + n)$ will not be convertible either.

In the 90's, new attempts to incorporate user-defined computations as rewrite rules were carried out. This resulted in the definition of the Calculus of Algebraic Constructions [6]. By introducing the correct rewriting rules, the calculus is able to identify terms like $x + S(y)$ and $S(x) + y$. Although quite powerful (CAC captures CIC [7]), this paradigm does not yet fulfill all needs, as it fails to identify open terms like $x + y$ and $y + x$.

Further steps in the direction of integrating first-order theories into the Calculus of Constructions are Stehr's Open Calculus of Constructions (OCC) [8] and Oury's Extensional Calculus of Constructions (ECC) [9]. OCC allows the use of an arbitrary equational theory in conversion. ECC can be seen as a particular case of OCC in which all provable equalities can be used in conversion, which can also be achieved by adding the extensionality and Streicher's axiom [10] to CIC, hence the name of this calculus. Unfortunately, strong normalization and decidability of type checking are lost in ECC and OCC.

In a preliminary work, we designed a new, rather restrictive framework, the Calculus of Congruent Constructions, which incorporates the congruence closure algorithm [11] in CC's conversion, while preserving the good properties of the calculus. We then defined the Calculus of Inductive Congruence Constructions (CCIC [12, 13]), which was our first answer to the problem of incorporating validity entailment of an equational first-order theory $\mathcal{T}$ in the computation mechanism of the Calculus of Inductive Constructions. This calculus partially solved the problem but contained too many ad-hoc restrictions to be implementable in practice. In particular, the procedure could not be invoked to solve goals occurring below eliminators.

Problem. The main question investigated in this paper is the incorporation of a general mechanism invoking a decision procedure for solving conversion-goals of the form $\Gamma \Rightarrow U = V$ in the Calculus of Inductive Constructions which uses the relevant information available from the current context Γ of the proof. This mechanism should be simple enough to be implementable in practice.

Theoretical contribution. Our theoretical contribution is the definition and the meta-theoretical study of a new version of the Calculus of Congruent Inductive Constructions (called CoqMT). Not only is the new formulation much simpler, but we succeeded in addition to remove most ad-hoc restrictions of CCIC. In particular, conversion now contains the entire equational theory $\mathcal{T}$. We show that we keep all the desired properties of such a calculus: subject reduction, strong normalization of reduction, logical consistency and decidability of type-checking.

Practical contribution. We implemented CoqMT, a new version of Coq based on the Calculus of Inductive Congruent Constructions. We also formally defined our calculus in Coq, and started its formal meta-theoretical study.[2]

[2] This development is available at `http://strub.nu/research/coqmt/`

We assume the reader familiar with typed λ-calculus [14] and rewriting [15]. *Due to the lack of space, we restrict our presentation to the Calculus of Constructions with two inductive types, natural numbers and lists depending on their size. The only embedded theory is the theory of Presburger arithmetic. This version captures all technical and theoretical problems encountered in the development of the full version of the calculus.*

From now on, $\mathcal{T}$ designates the first-order theory of Presburger arithmetic, built from the function symbols 0, S and $+$ and on the equality predicate $=$. By extension, we write $\mathcal{T}$ for the set of first-order terms belonging to the theory $\mathcal{T}$. If $\mathcal{E} = \{t_1 = u_1, \ldots, t_n = u_n\}$ is a finite set of equations in $\mathcal{T}$ (or $\mathcal{T}$-equations) and $t = u$ is a $\mathcal{T}$-equation, we write $\mathcal{T}, \mathcal{E} \vDash t = u$ if $t_1 = u_1 \wedge \cdots \wedge t_n = u_n \Rightarrow t = u$ is a valid formula in $\mathcal{T}$.

1.1 Syntax and notations

Definition 1 (Terms). *The set of* CoqMT *pseudo-terms is defined by:*

$$t, u, T, U, \ldots ::= s \in \{\star, \Box\} \mid x \in \mathcal{X} \mid f \in \Sigma \mid \bullet[t :^{e} T].\, u \mid u\, v \\ \mid \lambda[x :^{e?} U].\, v \mid \forall(x :^{e?} U).\, V \mid \mathrm{Elim}(t : Q)\{f_0, f_S\} \mid \doteq \mid \mathrm{refl}_T(t)$$

where $\Sigma = \{\mathbf{0}, \mathbf{S}, +, \mathbf{nil}, \mathbf{cons}, \mathbf{nat}, \mathbf{list}\}$*, e denotes an* extracted equation*, a notion to be explained later. The notation e? denotes either an extracted equation e or the symbol $\perp$. To ease the reading, we allow us to omit the $\perp$ symbol.*

Pseudo-terms of CoqMT are made of the usual terms of the Calculus of Constructions: *variables*, *sorts*, *abstraction* $\lambda[x : U].\, v$, *dependent product* $\forall(x : U).\, V$. Binders can be optionally decorated with an *extracted equation* e. For dealing with equalities and natural numbers, the following symbols are added: **nat** is the type of natural numbers, whereas **0**, **S** and $+$ are resp. the *zero*, *successor* and *addition* symbols. Similarly, $\doteq$ is the (dependent) equality predicate ($\doteq\ T\, t_1\, t_2$ or $t_1 \doteq_T t_2$ denotes that t_1 and t_2 are two equal terms of type T), while $\mathrm{refl}_T(t)$ denotes the proof by reflexivity of $t \doteq_T t$. Also, to illustrate the use of integrating Presburger arithmetic in CoqMT, we add symbols for dealing with dependent lists: **list** is the type of dependent lists, whereas **nil** and **cons** are its constructors. The construction $\mathrm{Elim}(t : Q)\{f_0, f_S\}$ denotes the usual recursor for natural numbers. Its semantic is detailed later when defining the reduction relation of our calculus.

A major novelty of CoqMT is the new symbol $\bullet[t :^{e} T].\, u$, called *extracted equation marker* or simply *equation marker*. It aims at carrying the equation e available for conversion - the terms t and T being present for typing purposes. Definitions to come ensure that this construction does not interfere with reduction: if substituting u in a term creates redexes, substituting $\bullet[t :^{e} T].\, u$ in the same term should create the same *kind* of redexes at the same positions.

We still need to clarify what is an *extracted equation*. As our calculus will incorporate the validity entailment of $\mathcal{T}$ in its conversion along with $\mathcal{T}$-equations extracted from the environment, we first need a process to convert CoqMT

terms into $\mathcal{T}$ terms: this is called *algebraisation*. For CoqMT terms containing only symbols relative to **nat** and variables, algebraisation is simply the process of currying symbols: from $\mathbf{S}\,x + \mathbf{0}$, we can extract the $\mathcal{T}$-term $S(x) + 0$ - note the font change. Unfortunately, due to substitution occurring in β-reduction, extracting only this kind of terms is not sufficient. Indeed, when substituting x by a arbitrary term t in the previous example, we first abstract t in the expression $\mathbf{S}\,t + \mathbf{0}$ by a fresh *variable* (say y), and then do the algebraisation. Here, we obtain the $\mathcal{T}$-term $S(y) + 0$ with y abstracting t.

Definition 2 (Algebraic context). *A context is said* algebraic *if it is of the form* $C_{\mathcal{A}} ::= \mathbf{0} \mid \mathbf{S}\,C_{\mathcal{A}} \mid \bullet_n + \bullet_p \mid \bullet_n$*, and not reduced to* $\bullet_n$*, where* $\bullet_n$ *denotes a hole with index* $n \in \mathbb{N}$*. By convention, two different holes in a context cannot have the same index.*

If $C_{\mathcal{A}}$ is an algebraic context whose maximal holes index is n, we write $C_{\mathcal{A}}[1 \leftarrow t_1, \ldots, n \leftarrow t_n]$ the term obtained by replacing the i^{th} hole of $C_{\mathcal{A}}$ by t_i. We call *context instantiation* the construct $[1 \leftarrow t_1, \ldots, n \leftarrow t_n]$, and denote by the letters $\mathcal{I}$, $\mathcal{J}$ *instantiation variables*. A *pre-algebraic* term is simply a term which can be obtained by instantiation of an algebraic context.

Definition 3 (Pre-algebraic term). *A pseudo-term* t *is* pre-algebraic *if it is of the form* $C_{\mathcal{A}}[1 \leftarrow t_1, \ldots, n \leftarrow t_n]$*. If none of the* t_i*'s are pre-algebraic, we say that* $C_{\mathcal{A}}$ *is the* maximal algebraic cap of t *and* $\{t_1, \ldots, t_n\}$ *are the* aliens *of* t*.*

For example, the term $(\mathbf{S}\,t) + u$ is pre-algebraic. We can take $\bullet_1 + \bullet_2$ as algebraic context with the instantiation $[1 \leftarrow \mathbf{S}\,t, 2 \leftarrow u]$. This context is not maximal. Assuming t and u not headed by a symbol of $\mathcal{T}$ (and hence not being pre-algebraic), $(\mathbf{S}\,\bullet_1) + \bullet_2$ is the maximal algebraic cap of $(\mathbf{S}\,t) + u$, and t, u are its two aliens.One can remark that a pseudo-term has exactly one maximal algebraic cap and the set of aliens is uniquely defined. Note that we consider the variables of a CoqMT term as aliens.

Definition 4 (Algebraisation). *For any term* t*, an* algebraisation *of* t *is a pair* $(C, \mathcal{I})$ *s.t.* $t = C[\mathcal{I}]$*. If* C *is the maximal algebraic cap of* t*, we say that* $(C, \mathcal{I})$ *is the* maximal algebraisation of t*.*

Definition 5 (Extracted equation). *An extracted equation is a* 4*-tuple* $(C_1, C_2, \mathcal{I}_1, \mathcal{I}_2)$ *(written* $(C_1 = C_2, \mathcal{I}_1, \mathcal{I}_2)$*) s.t.* C_1 *and* C_2 *are algebraic context and* $\mathcal{I}_1$*,* $\mathcal{I}_2$ *are context instantiations.*

For example, $(\bullet_1 + \bullet_2 = \mathbf{0}, [1 \leftarrow t, 2 \leftarrow u], [])$ is an extracted equation. By replacing holes with fresh variables and currying, we obtain the $\mathcal{T}$-equation $x + y = 0$ which can be used in a conversion check. Context instantiations allows us to related the extracted equations to CoqMT types: the extracted equation $(\bullet_1 + \bullet_2 = 0, [1 \leftarrow t, 2 \leftarrow u], [])$ will be extractable from a CoqMT term T if and only if T is of the form $(\bullet_1 + \bullet_2)[1 \leftarrow t, 2 \leftarrow u] \doteq \mathbf{0}[]$ i.e. of the form $t + u \doteq \mathbf{0}$.

When extracted equations are *pure*, i.e. the instantiations fill holes with variables only, we simplify our notations by giving the instantiated extracted equations instead of the 4-tuple. For example, we write $x + \mathbf{0} = \mathbf{S}\,\mathbf{0}$ for the extracted equation $(\bullet_1 + \mathbf{0} = \mathbf{S}\,\mathbf{0}, [1 \leftarrow x], [])$.

Definition 6 (Substitution). *We denote by $t\theta$ with $\theta = \{x \leftarrow w\}$ the term obtained by substituting in t all free occurrences of x by w - including the term appearing in the extracted equations:*

$$
\begin{array}{ll}
f\theta = f \quad s\theta = f \quad (\doteq \theta) = (\doteq) & x\theta = w \quad y\theta = y \text{ when } x \neq y \\
(\bullet[t :^e T].\, u)\theta = \bullet[t\theta :^{e\theta} T\theta].\, u\theta & (u\,v)\theta = (u\theta)\,(v\theta) \\
(\lambda[x :^{e?} U].\, v)\theta = \lambda[x :^{e?\theta} U\theta].\, v\theta & (\forall(x :^{e?} U).\, v)\theta = \forall(x :^{e?\theta} U\theta).\, v\theta \\
(\mathrm{Elim}(t : Q)\{f_0, f_S\})\theta = & (\mathrm{refl}_T(t))\theta = \mathrm{refl}_{T\theta}(t\theta) \\
\quad \mathrm{Elim}(t\theta : Q\theta)\{f_0\theta, f_S\theta\} & \\
(C_1 = C_2, \mathcal{I}, \mathcal{J})\theta = (C_1 = C_2, \mathcal{I}\theta, \mathcal{J}\theta) & \bot\,\theta = \bot \quad [i \leftarrow t_i]\theta = [i \leftarrow t_i\theta]
\end{array}
$$

Regarding variables capture, we here assume the Barendregt convention.

Notation 1. By $\{\vec{t_i}\}_{1 \leq i \leq n}$, or simply $\vec{t}$, we denote the terms $t_1\, t_2 \cdots t_n$. We write $T \to^{e?} U$ for $\forall(x :^{e?} T).\, U$ when x is not free in T. When using sequences, we write ϵ for the empty sequence and $x :: xs$ for the `cons` operation. The notation $[x_1, x_2, \ldots, x_n]$ denotes $x_1 :: x_2 :: \cdots :: \epsilon$.

1.2 Converting terms

Our conversion relation is split in two parts. The first part is a rewriting relation $\to_{\beta\iota\mathcal{T}}$, which includes β-reduction and the Martin-Löf recursor reduction for natural numbers, so called ι-reduction. Our reduction $\to_{\beta\iota\mathcal{T}}$ differs from the standard $\to_{\beta\iota}$ by its ability to operate modulo the theory $\mathcal{T}$. For example, the term $\mathrm{Elim}((x + (\mathbf{S}\,\mathbf{0})) + y : Q)\{f_0, f_S\}$, which is head-normal for $\to_{\beta\iota}$, is head-reducible for $\to_{\beta\iota\mathcal{T}}$. Actually, our definition of $\to_{\beta\iota\mathcal{T}}$ is such that $\mathrm{Elim}((x + (\mathbf{S}\,\mathbf{0})) + y : Q)\{f_0, f_S\}$ (resp. $\mathrm{Elim}(\mathbf{S}\,(x + y) : Q)\{f_0, f_S\}$) $\to_{\beta\iota\mathcal{T}}$-head-reduces (resp. $\to_{\beta\iota}$-head-reduces) to the same term $f_S\,(x + y)\ \mathrm{Elim}(x + y : Q)\{f_0, f_S\}$. The second part of our conversion relation is the extension of the validity entailment of $\mathcal{T}, E$ to the set of CoqMT pseudo-terms, where E is a set of extracted equations. We call this the *$\mathcal{T}$-congruence relation*. This $\mathcal{T}$-congruence will be parametrized by its ability to use the equations in E or not. We write $\equiv_E^\top$ for the $\mathcal{T}$-congruence having the ability to use the equations of E, and $\equiv_E^\bot$ for the other one. For example, if from E we can use the equation $x = (\mathbf{S}\,\mathbf{0})$, $\equiv_E^\top$ will be such that $x + x \equiv_E^\top (\mathbf{S}\,(\mathbf{S}\,\mathbf{0}))$, whereas $x + x$ and $\mathbf{S}\,(\mathbf{S}\,\mathbf{0})$ will not be convertible using $\equiv_E^\bot$.

We start with the definition of the reduction relation $\to_{\beta\iota\mathcal{T}}$. (From now on, for ease of notations, we write $\to$ for $\to_{\beta\iota\mathcal{T}}$) We achieve the reduction modulo by applying a normalization function to terms being in deconstruction position [16, 17]. For that, we introduce a function $\mathrm{norm}_\mathcal{T}$ normalizing first-order $\mathcal{T}$-terms as found in Shostak's method for combining decision procedures [18]:

1. $\mathcal{T} \vDash t = \mathrm{norm}_{\mathcal{T}}(t)$, 2. the variables of $\mathrm{norm}_{\mathcal{T}}(t)$ appear in t,
3. $\mathrm{norm}_{\mathcal{T}}$ is involutive, 4. sub-terms of a normalized term are normalized.

We lift the $\mathrm{norm}_{\mathcal{T}}$ function to pre-algebraic terms as follows:

Definition 7 ($\mathcal{T}$-normalization). *Let $t = C[\mathcal{I}]$ a* CoqMT *term with maximal algebraic context C, and $\mathcal{I} = [1 \leftarrow t_1, \ldots, n \leftarrow t_n]$. Let $\mathcal{J} = [1 \leftarrow x_1, \ldots, n \leftarrow x_n]$ where all the x_i's are pairwise different, and not free in the t_i's. The $\mathcal{T}$-normalization of t, written $\mathbf{norm}(t)$, is defined as:*

$$\mathbf{norm}(t) = \mathrm{norm}_{\mathcal{T}}(t[\mathcal{J}])\{x_1 \to t_1\} \cdots \{x_n \to t_n\}.$$

For example, assuming $\mathrm{norm}((x + S(0)) + y) = S(x + y)$, then $\mathbf{norm}((u + (\mathbf{S}\,\mathbf{0})) + u') = \mathbf{S}\,(u + u')$. We could have defined a stronger normalization function in which sub-expressions are abstracted by the same variable when they are in the same equivalence class of the conversion relation of the calculus. This would have lead to a mutual definition of normalization, rewriting and conversion, a possibility explored in an unpublished work with a heavy price. We now have all the ingredients for defining our reduction relation:

Definition 8 (Reduction). *The rewriting relation $\to$ is the smallest relation stable by context and substitution s.t.:*

(β) $(\bullet[\overrightarrow{w :^e W}].\, \lambda[x : U].\, v)\, u \to \bullet[\overrightarrow{w :^e W}].\, v\{x \leftarrow u\}$

$(\beta\text{-}\mathcal{E})$ $(\bullet[\overrightarrow{w :^e W}].\, \lambda[x :^{e'} U].\, v)\, u \to \bullet[\overrightarrow{w :^e W}].\, \bullet[u :^{e'} U].\, v\{x \leftarrow u\}$

(ι_0) $\mathrm{Elim}(t : Q)\{f_0, f_S\} \to f_0$ *if t is pre-algebraic, and* $\mathbf{norm}(t) = \mathbf{0}$

(ι_S) $\mathrm{Elim}(t : Q)\{f_0, f_S\} \to f_S\, v\, \mathrm{Elim}(v : Q)\{f_0, f_S\}$ *if t is pre-algebraic, and* $\mathbf{norm}(t) = \mathbf{S}\, v$

Rule (β) (resp. $(\beta - \mathcal{E})$ could have been split in two more basic actions:

$$\begin{aligned} (\bullet[\overrightarrow{w :^e W}].\, \lambda[x : U].\, v)\, u &\to_{\bullet} \bullet[\overrightarrow{w :^e W}].\, ((\lambda[x : U].\, v)\, u) \quad (1) \\ &\to_{\beta} \bullet[\overrightarrow{w :^e W}].\, v\{x \leftarrow u\} \quad (2) \end{aligned}$$

where reduction (1) is the action of moving the bullet expression hiding the β-redex, whereas reduction (2) is the standard β-reduction. This remark applies to $(\beta\text{-}\mathcal{E})$ also.

Rule $(\beta\text{-}\mathcal{E})$ has an additional role: the introduction of an equation marker as is clear for its instance $(\lambda[x :^e U].\, v)\, u \to \bullet[u :^e U].\, v\{x \leftarrow u\}$. Equation markers are essential for the subject reduction property: consider for example the term $(\lambda[p :^{x=\mathbf{0}} T].\, \mathrm{refl}(x))\, t$. Being a proof of reflexivity, $\mathrm{refl}(x)$ has type $x \doteq x$. Using the extracted equation $x = \mathbf{0}$, our conversion rule should allow us to derive $[x : \mathbf{nat}][p :^{x=\mathbf{0}} T] \vdash \mathrm{refl}(x) : x \doteq \mathbf{0}$, and hence $[x : \mathbf{nat}] \vdash (\lambda[x :^{x=0} T].\, \mathrm{refl}(x))\, t : x \doteq \mathbf{0}$ as well. Without the introduction of equation markers, β-reduction would yield $(\lambda[x :^{x=\mathbf{0}} T].\, \mathrm{refl}(x))\, t \to_{\beta} \mathrm{refl}(x)$ of type $x \doteq x$ which is not convertible to $x \doteq \mathbf{0}$ anymore since the equation $x = \mathbf{0}$ has been lost.

Adding the equation marker ($\bullet[t :^{x=\mathbf{0}} T].\_$) at root position solves the problem by reintroducing the extracted equation $x = \mathbf{0}$.

This problem already appeared in [13], and was solved by forbidding the application of annotated λ-abstractions to other terms. At that time, we thought the problem could be by-passed by locking such problematic β-redexes. Although our calculus is more general than a calculus where problematic β-redexes are locked (the latter can be easily encoded in the former), its study is much easier: locking β-redexes leads to normal terms containing β-redexes, and hence the standard proof of consistency based on cut elimination does not apply anymore.

The two last rules deal with reduction of Martin-Löf like recursors. Usual reduction rules for elimination of **nat** are:

$$\begin{array}{l}\mathrm{Elim}(\mathbf{0} : Q)\{f_0, f_S\} \ \ \to f_0\\ \mathrm{Elim}(\mathbf{S}\, t : Q)\{f_0, f_S\} \to f_S\, t\ \mathrm{Elim}(t : Q)\{f_0, f_S\}\end{array}$$

As said, reduction is done modulo the theory $\mathcal{T}$ through the use of **norm**. For example, the term $\mathrm{Elim}((x + (\mathbf{S}\,\mathbf{0})) + y : Q)\{f_0, f_S\}$ which is normal for the standard ι-reduction will reduce to f_0 using ι_0, as $\mathbf{norm}((x + (\mathbf{S}\,\mathbf{0})) + y) = \mathbf{S}(x + y)$. Reduction also contains the recursor rules for lists which are identical to the corresponding ones in the Calculus of Inductive Constructions, and do not play any critical role in our definition.

Notation 2. We write $\xrightarrow{\leq}$, $\xrightarrow{*}$, $\xleftrightarrow{*}$ for resp. the reflexive, reflexive-transitive and equivalence closure of $\to$. We write $t{\to_\downarrow}u$ if t has a unique normal form equal to u.

We move now to the definition of our conversion relation. We start explaining how $\mathcal{T}$-validity entailment makes use of pre-algebraic terms and extracted equations, via the notion of $(\mathcal{T}, E)$-consequence. Let us assume the following two extracted equations of E: i) $(\bullet_1 + \bullet_2 = \mathbf{S}\,\mathbf{0}, [1 \leftarrow t, 2 \leftarrow u], [])$, and ii) $(\mathbf{S}\,\bullet_1 = \mathbf{S}\,\mathbf{0}, [1 \leftarrow t'], [])$; with $t \xleftrightarrow{*} t'$. Then, for any u' s.t. $u \xleftrightarrow{*} u'$, we say that $x + u' \doteq \mathbf{S}\,x$ is a $(\mathcal{T}, E)$-consequence modulo $\xleftrightarrow{*}$ as the entailment $\mathcal{T}, z_1 + z_2 = 1, S(z_3) = 1, \mathcal{V} \vDash x + z_4 = S(x)$ holds, taking $\mathcal{V} = \{z_1 = z_3, z_2 = z_4\}$:

- $z_1 + z_2 = 1$ and $S(z_3) = 1$ are the two extracted equations of E where holes are replaced by pairwise disjoint fresh variables. Hence, z_1 (resp. z_2 and z_3) abstracts t (resp. u and t'),
- from $t \xleftrightarrow{*} t'$ (resp. $u \xleftrightarrow{*} u'$), we add the equation $z_1 = z_3$ (resp. $z_2 = z_4$) to the set of available equations, via the set $\mathcal{V}$.
- $x + z_4 = S(x)$ (the conclusion of the $\mathcal{T}$-entailment) is the algebraisation of $x + u' = \mathbf{S}\,x$.

We now describe formally the notion of $(\mathcal{T}, E)$-consequence.

Definition 9. *Let $\{C_i\}_{1\leq i\leq n}$ a set of pre-algebraic contexts, $\{\mathcal{I}_i\}_{1\leq i\leq n}$ a set of context instantiations and $\mathcal{R}$ a binary relation on* CoqMT *terms. For any $i \in \{1..n\}$, k, let $t_{i,k}$ the term associated to the k^{th} hole by $\mathcal{I}_i$.*

Let $\{\mathcal{J}_i\}_{1\leq i\leq n}$ be a set of instantiations and $\mathcal{V}$ a set of equations between variables. We say that $\{\mathcal{J}_i\}_i$ abstracts *$\{(C_i, \mathcal{I}_i)\}_i$* accordingly to *$\mathcal{V}$* and *$\mathcal{R}$ if:*

1. *for any* $i \in \{1..n\}$, k, $\mathcal{J}_i$ *associates a fresh variable* $x_{i,k}$ *to the* k^{th} *hole of* C_i,
2. $(x_{i,p} = x_{j,q}) \in \mathcal{V}$ *implies* $t_{i,p}\ \mathcal{R}\ t_{j,q}$

Definition 10 ($(\mathcal{T}, E)$-consequence). *Let* $E = [(C_i^l = C_i^r, \mathcal{I}_i^l, \mathcal{I}_i^r) \mid 1 \leq i \leq n]$ *be a sequence of extracted equations, and* t, u *two* CoqMT *terms of the form* $C_0^l[\mathcal{I}_0^l]$ *and* $C_0^r[\mathcal{I}_0^r]$ *respectively. Let* $\mathcal{R}$ *be any binary relation on* CoqMT *terms.*

We say that $(t = u)$ *is a* $(\mathcal{T}, E)$*-consequence modulo* $\mathcal{R}$ *if there exists a set* $\{\mathcal{J}_i^\alpha\}_{i,\alpha}$ *of context instantiations and a set* $\mathcal{V}$ *of equations between variables s.t.*

1. $\mathcal{T}, \{C_i^l[\mathcal{J}_i^l] = C_i^r[\mathcal{J}_i^r]\}, \mathcal{V} \vDash C_0^l[\mathcal{J}_0^l] = C_0^r[\mathcal{J}_0^r]$,
2. $\{\mathcal{J}_i^\alpha\}_{i,\alpha}$ *abstracts* $\{(C_i^\alpha, \mathcal{I}_i^\alpha)\}_{i,\alpha}$ *accordingly to* $\mathcal{V}$ *and* $\mathcal{R}$.

We introduce two variants of conversion: a *strong* one which can use the equations of E, and a *weak* one which cannot. Doing this is crucial for the logical consistency proof of our calculus in the presence of strong recursors (i.e. construction of types by induction). Assume the function $f(n) := \mathrm{Elim}(n : Q)\{\mathbf{nat}, \lambda[\_.\mathbf{nat}].\, \lambda[T : \star].\, T \to \mathbf{nat}\}$, i.e. a function s.t. $f(n) = \overbrace{\mathbf{nat} \to \cdots \to \mathbf{nat}}^{n \text{ times}}$. If we allow the use of any equation in the conversion of n of $f(n)$, for example the equation $\mathbf{0} \doteq \mathbf{S}\,\mathbf{0}$, then $f(0)$ would be convertible to $f(1)$ which respectively $\to$-reduce to **nat** and $\mathbf{nat} \to \mathbf{nat}$. Having **nat** convertible to $\mathbf{nat} \to \mathbf{nat}$ would then allows us to type-check the term $(\lambda[x : \mathbf{nat}].\, x\, x)\, (\lambda[x : \mathbf{nat}].\, x\, x)$, which is not strongly-normalizing.

Restricting conversion under recursors is not enough: let $A = (\lambda[x : \mathbf{nat}].\, f(x))\, t$, in which f is defined as above. If we allow t to be convertible to $\mathbf{0}$ and $\mathbf{S}\,\mathbf{0}$, since $A \to_\beta f(t)$, we are back to the previous example. We called *strong* such problematic terms. Of course, we want a syntactic characterization of non-strong terms. The following lemma gives one which is convenient for its use in the context of dependent data types:

Lemma 1. *Terms headed by constructor and inductive symbols (hence* **0**, **S**, **nil**, **cons**, **list***) are non-strong. We call them* weak.

We now define the $\mathcal{T}$-congruence $\equiv_E^b$:

Definition 11. *The* $\mathcal{T}$*-congruence* $\equiv_E^b$, $b \in \{\top, \bot\}$ *is given in Figure 1.*

The $\mathcal{T}$-congruence is defined as follows. First, our notion of $(\mathcal{T}, E)$-consequence is included through the [Ded] rules. In the case of a weak conversion, the [$\mathrm{Ded}_\bot$] rule does not use extracted equations, and uses the empty relation for aliens comparison. The rules [$\bullet$], [Lam], [Elim] and [Prod] are congruence rules. Of course, when crossing a binder annotated with an extracted equation, the set of available equations must be updated (this was already the case in [13]). Note also that in the definition of conversion for recursors, the term in deconstructing position can only be converted using weak conversion. There

$$[\textsc{Refl}]\ \frac{}{t \equiv_E^b t} \qquad [\textsc{Trans}]\ \frac{t_1 \equiv_E^b t_2 \quad t_2 \equiv_E^b t_3}{t_1 \equiv_E^b t_3}$$

$$[\textsc{App}_{\mathcal{W}}]\ \frac{\begin{array}{c}t, u \text{ are weak terms} \\ t \equiv_E^\top u \qquad \forall i, t_i \equiv_E^\top u_i\end{array}}{t\, t_1 \cdots t_n \equiv_E^\top u\, u_1 \cdots u_n} \qquad [\textsc{App}_{\mathcal{S}}]\ \frac{\forall i, t_i \equiv_E^\bot u_i}{t_1 \cdots t_n \equiv_E^b u_1 \cdots u_n}$$

$$[\textsc{Lam}]\ \frac{U \equiv_E^b U' \quad v \equiv_E^b v'}{\lambda[x : U].\, v \equiv_E^b \lambda[x : U'].\, v'} \qquad [\textsc{Prod}]\ \frac{U \equiv_E^b U' \quad v \equiv_E^b v'}{\forall(x : U).\, v \equiv_E^b \forall(x : U').\, v'}$$

$$[\textsc{Lam-}\mathcal{E}]\ \frac{U \equiv_E^b U' \quad v \equiv_{e::E}^b v' \quad e \equiv_E^b e'}{\lambda[x :^e U].\, v \equiv_E^b \lambda[x :^{e'} U'].\, v'} \qquad [\textsc{Prod-}\mathcal{E}]\ \frac{U \equiv_E^b U' \quad v \equiv_{e::E}^b v' \quad e \equiv_E^b e'}{\forall(x :^e U).\, v \equiv_E^b \forall(x :^{e'} U').\, v'}$$

$$[\bullet]\ \frac{\begin{array}{c}t \equiv_E^b t' \quad T \equiv_E^b T' \\ U \equiv_{e::E}^b U' \quad e \equiv_E^b e'\end{array}}{\bullet[t :^e T].\, U \equiv_E^b \bullet[t' :^{e'} T'].\, U'} \qquad [\textsc{Eq}]\ \frac{\begin{array}{c}\forall i,\, t_i \equiv_E^b u_i \quad \forall i,\, t_i' \equiv_E^b u_i' \\ e_1 = (C = C', [i \leftarrow t_i], [i \leftarrow t_i']) \\ e_2 = (C = C', [i \leftarrow u_i], [i \leftarrow u_i'])\end{array}}{e_1 \equiv_E^b e_2}$$

$$[\textsc{Ded}_\top]\ \frac{\begin{array}{c}t = u \text{ is a } (\mathcal{T}, E)\text{-consequence} \\ \text{modulo } \xleftrightarrow{*} \cdot \equiv_E^b \cdot \xleftrightarrow{*}\end{array}}{t \equiv_E^\top u} \qquad [\textsc{Ded}_\bot]\ \frac{\begin{array}{c}t = u \text{ is a } (\mathcal{T}, \emptyset)\text{-consequence} \\ \text{modulo the empty relation}\end{array}}{t \equiv_E^\bot u}$$

$$[\textsc{Elim}]\ \frac{t \equiv_E^\bot t' \quad Q \equiv_E^b Q' \quad f_0 \equiv_E^b f_0' \quad f_S \equiv_E^b f_S'}{\mathrm{Elim}(t : Q)\{\overrightarrow{f}\} \equiv_E^b \mathrm{Elim}(t' : Q')\{\overrightarrow{f'}\}}$$

Fig. 1: CoqMT $\mathcal{T}$-congruence relation

are two rules for application. As explained, applications headed with a weak term can be converted using strong conversion. Otherwise, weak conversion must be used. Finally, the rule [Eq] is used to convert extracted equations: two extracted equations are convertible if they contain the same algebraic cap and have pairwise convertible aliens.

The conversion $\sim_E$ of CoqMT, w.r.t. a set E of extracted equations, is simply defined as $(\xleftrightarrow{*} \cdot \equiv_E^\top \cdot \xleftrightarrow{*})$.

1.3 CoqMT typing

Contrary to our previous calculus [13], extracted equations are now directly accessible via annotations. Although this greatly simplifies the definition of the calculus by removing the CCIC complex notion of extraction, one must verify some consistency properties between the extracted equations and the terms which are annotated with. For example, from the term $x \doteq \mathbf{S}\, t$, one wants $x = \mathbf{S}\, y$ to be a valid extractable equation (with y abstracting t), but not $x = \mathbf{0}$. Allowing the extraction of $x = \mathbf{0}$ from $x \doteq \mathbf{S}\, t$ would allow users to prove $x \doteq \mathbf{S}\, t$ and $x \doteq \mathbf{0}$ without the use of any assumptions, thus leading to a non-consistent system.

Definition 12 (E-extractability). *Let $(C_1 = C_2, \mathcal{I}, \mathcal{J})$ be an extracted equation and T a term. We say that e* is extractable from *T if T is convertible, using $\sim_E$, to $C_1[\mathcal{I}] \doteq C_2[\mathcal{J}]$.*

Before defining our typing judgment, we are left to define our typing environments. As usual, typing environments bind variables to types, but with two modifications. First, as for λ-expressions and dependent products, bindings can be optionally annotated with extracted equations. Second, typing environments will also contain equation markers.

Definition 13 (Typing environment). *The set of typing environments is defined by $\Gamma, \Delta ::= \epsilon \mid \Gamma, [x :^{e?} T] \mid \Gamma, \bullet[t :^{e?} T]$.*

Definition 14 (Typing judgement). *For any environment Γ, let $\sim_\Gamma$ defined as $\sim_E$ where E is the set of all extractable equations appearing in Γ. Typing rules of* CoqMT *are given in Figure 2.*

The rules for $\doteq$ and **nat** symbols are identical to the ones of CIC - for lack of space, we omit the rules for lists. Several modifications are made to the pure Calculus of Constructions rules. First, when adding to a typing environment Γ a new binding annotated by an extracted equation e (rules [Weak], [Weak-$\bullet$] and [Var]), we check that e is indeed extractable, hence making sure that we do not extract inconsistent equations from a consistent typing environment.

Moreover, the rule for application has been duplicated, as for β-reduction. For the two rules, we allow the left hand-side to have a product type modulo the presence of equation markers. This is for the non-interference of reduction as discussed before and is very similar to the equation markers we found in the (β) and $(\beta\text{-}\mathcal{E})$ in the definition of $\rightarrow$ rules. The second one, [App-$\mathcal{E}$], handles the case where a term annotated by an equation is applied. As for the $(\beta\text{-}\mathcal{E})$ rule which handles the case where an extracted equation annotating a λ-abstraction has disappeared, [App-$\mathcal{E}$] handles the case of for dependent products with the very same technique.

We also add two rules for typing equation markers. One is a weakening style rule and allows the user to add extracted equations to the environment. The second allows markers to be moved from terms and types to the environment.

The conversion rule is of course updated to use the relation $\sim_\Gamma$.

2 Meta-theoretical properties

The meta-theory of our calculus is carried out in a similar way as that of CC or CIC. As usual, the logical consistency is proved in three steps. First, we prove the *subject reduction* property stating that the reduction relation is correct w.r.t. typing: if $\Gamma \vdash t : T$ and $t \rightarrow t'$, then $\Gamma \vdash t' : T$. In the second step, we prove that the reduction relation is strongly normalizing for well-typed terms. In the last step, we prove that there exists a term which is non-typable in the empty environment.

$$[\textsc{Sort}]\ \frac{}{\vdash \star : \square} \qquad [\textsc{Var}]\ \frac{\Gamma \vdash T : s \qquad x \notin \mathrm{dom}(\Gamma) \qquad e? \text{ is extractable from } T}{\Gamma, [x :^{e?} T] \vdash x : T}$$

$$[\textsc{Weak}]\ \frac{x \notin \mathrm{dom}(\Gamma) \qquad e? \text{ is extractable from } V \qquad \Gamma \vdash t : T \qquad \Gamma \vdash V : s}{\Gamma, [x :^{e?} V] \vdash t : T} \qquad [\textsc{Weak-}\bullet]\ \frac{x \notin \mathrm{dom}(\Gamma) \qquad e \text{ is extractable from } T \qquad \Gamma \vdash t : T \qquad \Gamma \vdash u : U}{\Gamma, \bullet[t :^{e} T] \vdash u : U}$$

$$[\textsc{App}]\ \frac{\Gamma \vdash u : U \qquad \Gamma \vdash t : \bullet[\overrightarrow{t :^{e} T}].\, \forall (x : U).\, V}{\Gamma \vdash t\, u : \bullet[\overrightarrow{t :^{e} T}].\, V\{x \leftarrow u\}} \qquad [\textsc{App-}\mathcal{E}]\ \frac{\Gamma \vdash u : U \qquad \Gamma \vdash t : \bullet[\overrightarrow{t :^{e} T}].\, \forall (x :^{e} U).\, V}{\Gamma \vdash t\, u : \bullet[\overrightarrow{t :^{e} T}].\, \bullet[u :^{e} U].\, V\{x \leftarrow u\}}$$

$$[\textsc{Lam}]\ \frac{\Gamma, [x :^{e?} U] \vdash v : V \qquad \Gamma \vdash \forall (x :^{e?} U).\, V : s}{\Gamma \vdash \lambda [x :^{e?} U].\, v : \forall (x :^{e?} U).\, V} \qquad [\textsc{Prod}]\ \frac{\Gamma \vdash U : s_1 \qquad \Gamma, [x :^{e?} U] \vdash V : s_2}{\Gamma \vdash \forall (x :^{e?} U).\, V : s_2}$$

$$[\bullet]\ \frac{e \text{ is extractable from } T \qquad \Gamma, \bullet[t :^{e} T] \vdash u : U \qquad \Gamma \vdash t : T}{\Gamma \vdash \bullet[t :^{e} T].\, u : \bullet[t :^{e} T].\, U} \qquad [\textsc{Conv}]\ \frac{\Gamma \vdash t : T \qquad \Gamma \vdash U : s \qquad T \sim_{\Gamma} U}{\Gamma \vdash t : U}$$

$$[\mathbf{0}]\ \frac{}{\mathbf{0} : \mathbf{nat}} \qquad [\mathbf{S}]\ \frac{}{\mathbf{S} : \mathbf{nat} \to \mathbf{nat}} \qquad [+]\ \frac{}{+ : \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat}} \qquad [\mathbf{nat}]\ \frac{}{\mathbf{nat} : \star}$$

$$[\textsc{Refl}]\ \frac{\Gamma \vdash t : T}{\Gamma \vdash \mathrm{refl}_T(t) : t \doteq_T t} \qquad [\textsc{Eq}]\ \frac{}{\Gamma \vdash \doteq : \forall (T : \star).\, T \to T \to \star}$$

$$[\textsc{Elim}]\ \frac{\Gamma \vdash t : \mathbf{nat} \qquad \Gamma, [x : \mathbf{nat}] \vdash Q : s \qquad \Gamma \vdash f_0 : Q\{x \leftarrow \mathbf{0}\} \qquad \Gamma \vdash f_S : \forall (y : \mathbf{nat}).\, Q\{x \leftarrow y\} \to Q\{x \leftarrow \mathbf{S}\, y\}}{\mathrm{Elim}(t : \lambda [x : \mathbf{nat}].\, Q)\{f_0, f_S\} : Q\{x \leftarrow t\}}$$

Fig. 2: CoqMT typing relation

2.1 Subject reduction

The subject reduction proof requires the following property:

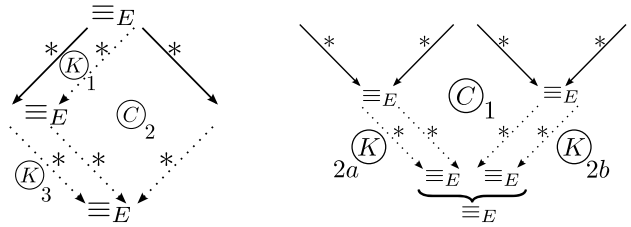
$$\forall (x :^{e} U).\, V \sim_E \forall (x :^{e'} U').\, V' \Rightarrow U \sim_E U' \wedge V \sim_{e::E} \wedge e \sim_E e'$$

called *product compatibility* [19]. Usually, conversion coincides with the closure by equivalence of reduction, and product compatibility is then an immediate consequence of the confluence property of reduction. In CCIC, product compatibility was obtained via a complex sequence of definitions and lemmas. It is much simpler here. The proof is structured as follows.

We first prove confluence of $\to$. The proof follows the method of Tait for β-reduction [20]: we define a parallel reduction $\Rightarrow$ s.t. $(\to^*) = (\Rightarrow^*)$, and prove its confluence as in Tait's proof. Let us give the definition for the case of (ι_0) reduction: $\mathrm{Elim}(t : Q)\{f_0, f_S\} \Rightarrow f_0'$ if $t \Rightarrow t'$, $f_0 \Rightarrow f_0'$ and $\mathbf{norm}(t') = \mathbf{0}$. We then prove the key property of $\mathcal{T}$-congruence, called *coherence* in [21]:

Lemma 2. *For any terms t, u, and set of extracted equation E, if $t \equiv_E^b u$ and $t \to t'$, then there exists a term u' s.t. $u \xrightarrow{\leq} u'$ and $t' \equiv_E^b u'$.*

Such a property is crucially based on the fact that our reduction is done modulo the theory. For example, the term $\text{Elim}((x + (\mathbf{S}\,\mathbf{0})) + y : Q)\{f_0, f_S\}$ is convertible to $\text{Elim}(\mathbf{S}\,(x + y) : Q)\{f_0, f_S\}$ and reducible using the (ι_S) rule. Thanks to the use of the **norm** function, we can conclude that $\sim_E$ is equal to $\xrightarrow{*} \cdot \equiv_E^\top \cdot \xleftarrow{*}$ (See Figure 3a), and then that $\sim_E$ is transitive (i.e. $\xrightarrow{*} \cdot \equiv_E^\top \cdot \xleftarrow{*}$ is transitive - see Figure 3b). Product compatibility for $\sim_E^*$ is then reduced to the product compatibility of $\sim_E$, which itself is reduced to the product compatibility of $\to$ and $\equiv_E^b$, two properties which are immediate.



(a) Key lemma (b) Transitivity

Fig. 3: Transitivity of $\to$ - (C) is application of confluence, and (K) of our key lemma

2.2 Strong normalization

In [13], we prove strong normalization of CCIC by an embedding into a variation of the Calculus of Algebraic Constructions [6, 7]. Due to the use of reduction modulo, this is no more possible. Proof of strong normalization of $\to$ is now obtained by adapting the proof for CIC to our calculus, which is based on Coquand and Gallier's extension to the Calculus of Constructions of Tait and Girard's computability predicate technique. The idea is to define an interpretation for each type and to prove that each well-typed term belongs to the interpretation of its type. The main difficulty is to define the interpretation for type constructed by induction. For the case of natural number, the CIC interpretation of $\text{Elim}(t : Q)\{f_0, f_S\}$ is

$$\begin{array}{ll} \text{the one of } f_0 & \text{if } t{\to_\downarrow}\mathbf{0} \\ \text{the one of } f_S\,u\ \text{Elim}(u : Q)\{f0, f_S\} & \text{if } t{\to_\downarrow}\mathbf{S}\,u \\ \text{an arbitrary value otherwise} & \end{array}$$

We simply adapt this definition by modifying the condition $t{\to_\downarrow}\mathbf{0}$ (resp. $t{\to_\downarrow}\mathbf{S}\,u$) by $t{\to_\downarrow}u$ with $\mathbf{norm}(u) = \mathbf{0}$ (resp. $t{\to_\downarrow}v$ with $\mathbf{norm}(v) = \mathbf{S}\,u$). The rest

of the proof is the same. The interpretation of the equation marker $\bullet[t :^e T].U$ is that of U. From strong normalization, we then deduce consistency as usual, by proving that there is no normal proof of false in the empty environment:

Theorem 1. *If $\Gamma \vdash t : T$, then t is strongly normalizable. Hence, there is no proof of $\mathbf{0} \doteq \mathbf{S}\,\mathbf{0}$ in the empty environment.*

Here, the choice of $\mathbf{0} \doteq \mathbf{S}\,\mathbf{0}$ follows from the fact that $\neg(\mathbf{0} \sim_\epsilon \mathbf{S}\,\mathbf{0})$. (Remember that ϵ is the empty sequence)

2.3 Decidability

Decidability is split in two parts. First, assuming decidability of the conversion relation, we shows decidability of the typing judgement. We define a syntax oriented (hence decidable) relation $\Gamma \vdash_i t : T$ s.t. if $\Gamma \vdash t : T$ then $\Gamma \vdash_i t : T'$ with $T \sim^*_\Gamma T'$. The definition of $\vdash_i$ is usually done by removing the conversion rule from [Conv] and integrating it in the rules for application (See Figure 4). All this is routine.

$$[\textsc{App}]\ \frac{\Gamma \vdash u : U' \qquad U \sim_\Gamma U' \qquad \Gamma \vdash t : \bullet[\overrightarrow{t :^e T}].\forall(x : U).V}{\Gamma \vdash t\,u : \bullet[\overrightarrow{t :^e T}].V\{x \leftarrow u\}} \qquad [\textsc{App-}\mathcal{E}]\ \frac{\Gamma \vdash u : U' \qquad U \sim_\Gamma U' \qquad \Gamma \vdash t : \bullet[\overrightarrow{t :^e T}].\forall(x :^e U).V}{\Gamma \vdash t\,u : \bullet[\overrightarrow{t :^e T}].\bullet[u :^e U].V\{x \leftarrow u\}}$$

Fig. 4: Application rule for $\vdash_i$

Second, we consider convertibility of well-formed (hence strongly normalizing) terms. We first slightly modify the $\mathcal{T}$-congruence by i) requiring maximal algebraic cap in rule [$\textsc{Ded}_b$], and ii) restrict the [App] rules to non pre-algebraic terms. This does not change the obtained relation, but allows us to remove [Trans], hence obtaining a syntax directed definition. (On the other hand, our initial formulation eased the meta-theoretical study) Finally, using again the key lemma used for subject reduction, we can prove that CoqMT conversion can be decomposed in a reduction phase followed by a $\mathcal{T}$-congruence phase:

Lemma 3. *Let t and u be two strongly normalized terms and E a set of extracted equations composed only of strongly normalized terms. Then $t \sim_E u$ if and only if $t_\downarrow \equiv^b_{E_\downarrow} u_\downarrow$ - where $w_\downarrow$ is the normal form of w*

Theorem 2. *The conversion $\sim$ is decidable on strongly normalizing terms.*

3 Implementation

A new version of Coq based on this work, called CoqMT, is available on the author's website[3]. CoqMT allows a user to dynamically load any decision

[3] `http://pierre-yves.strub.nu/research/coqmt/`

procedure for a first-order theory in the conversion of the system. It does not implement yet the mechanism of equations extraction. Full commented examples can be found in the source release of CoqMT (*/test-suite/dp/**).

4 Conclusions and future work

We have defined a new version of the Calculus of Congruent Inductive Constructions, called CoqMT, allowing the use of decision procedures in its conversion rule.

We drastically simplified the definition of CCIC, notably by removing almost all ad-hoc restrictions introduced in [13]. In particular, based on this, a formal proof of CoqMT in Coq is in development. As of today, an implementation of CoqMT allowing the use of decision procedures in the conversion rule is available. The implementation of the extraction mechanism is to be released soon. More details, along with all sources, can be found on the author's page. We now discuss interesting extension of this work.

We expect to remove the presence of equation markers in types by the use of sub-typing. In our work, non-interference of equation markers has only be addressed, in an ad-hoc manner, for β-reduction and typing of application. Extending this ad-hoc method to all constructions of the calculus would lead to an over-complicated definition. Introducing sub-typing s.t. $\bullet[t :^e T].\, A$ is a sub-type of A could lead to a calculus where a term of type $\forall(x : A).\, B$ is applicable to a term of type $\bullet[t :^e T].\, A$ for free.

Reduction modulo theory $\mathcal{T}$ does not use any extracted equation so that inconsistencies at the extracted equations level do not break logical consistency. A work-around would be to only use, during reduction, a subset of extracted equations which is consistent by construction. Although the idea is quite straightforward, due to the high dependency between weak and strong conversion, reduction modulo and consistency of extracted equation, we have not yet succeeded to obtain a workable definition. We hope solving this problem in a near future.

The link between first order theories and Coq symbols is a one-to-one mapping: each first-order symbol is mapped to a Coq constructor. We want to add a notion of view to break this restriction. This would allow e.g. to map Presburger arithmetic to a Coq binary representation of natural numbers. This could be extended to view dependent data-types as first order data-types with constraints. For example, assuming we embed the first-order theories of lists along with Presburger arithmetic in CoqMT, Coq dependent lists could be viewed by the theories as standard lists plus arithmetical constraints on their lengths.

Finally, we want to extend the extraction from equations to arbitrary first-order propositions (like ordering on natural numbers), or even add the ability to extract first-order formulas.

Acknowledgment: we thank Gilles Barthe and Jean-Pierre Jouannaud for useful discussions.

References

1. Coq-Development-Team: The Coq Proof Assistant Reference Manual - Version 8.2. INRIA, INRIA Rocquencourt, France. (2009) At URL `http://coq.inria.fr/`.
2. Coquand, T., Huet, G.: The Calculus of Constructions. Information and Computation **76**(2-3) (1988) 95–120
3. Coquand, T., Paulin-Mohring, C.: Inductively defined types. In Martin-Löf, Mints, G., eds.: Colog'-88, International Conference on Computer Logic. Volume 417 of LNCS., Springer-Verlag (1990) 50–66
4. Werner, B.: Une Théorie des Constructions Inductives. PhD thesis, University of Paris VII (1994)
5. Giménez, E.: Structural recursive definitions in type theory. In: Proceedings of ICALP'98. Volume 1443 of LNCS. (1998) 397–408
6. Blanqui, F.: Definitions by rewriting in the calculus of constructions. Mathematical Structures in Computer Science **15**(1) (2005) 37–92 Journal version of LICS'01.
7. Blanqui, F.: Inductive types in the calculus of algebraic constructions. Fundamenta Informaticae **65**(1-2) (2005) 61–86 Journal version of TLCA'03.
8. Stehr, M.: The Open Calculus of Constructions: An equational type theory with dependent types for programming, specification, and interactive theorem proving (part I and II). Fundamenta Informaticae **68** (2005)
9. Oury, N.: Extensionality in the calculus of constructions. In: Proceedings 18th TPHOL, Oxford, UK. LNCS 3603. (2005) 278–293
10. Hofmann, M., Streicher, T.: The groupoid model refutes uniqueness of identity proofs. In: LICS, IEEE Computer Society (1994) 208–212
11. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. J. ACM **27**(2) (1980) 356–364
12. Blanqui, F., Jouannaud, J., Strub, P.: Building decision procedures in the calculus of inductive constructions. In: Proceedings 16th CSL 2007, Lausanne, Switzerland. LNCS 4646. (2007) 328–342
13. Strub, P.Y.: Type Theory and Decision Procedures. PhD thesis, École Polytechnique (2008)
14. Barendregt, H.P.: Lambda calculi with types. (1992) 117–309
15. Dershowitz, N., Jouannaud, J.P.: Rewrite systems. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B). (1990) 243–320
16. Courtieu, P.: Normalized types. In Fribourg, L., ed.: CSL. Volume 2142 of Lecture Notes in Computer Science., Springer (2001) 554–569
17. Petcher, A., Stump, A.: Deciding joinability modulo ground equations in operational type theory. In: Proof Search in Type Theories (PSTT). (2009)
18. Shostak, R.E.: An efficient decision procedure for arithmetic with function symbols. J. of the Association for Computing Machinery **26**(2) (1979) 351–360
19. Barbanera, F., Fernández, M., Geuvers, H.: Modularity of strong normalization and confluence in the algebraic-lambda-cube. In: LICS, IEEE Computer Society (1994) 406–415
20. Barendregt, H.P.: The Lambda Calculus. Its Syntax and Semantics. North-Holland (1984) revised edition.
21. Jouannaud, J.P., Kirchner, H.: Completion of a set of rules modulo a set of equations. SIAM J. Comput. **15**(4) (1986) 1155–1194